

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

APPLICATION FOR LETTERS PATENT

**A SYSTEM AND METHODS  
FOR  
GENERATING AND MANAGING FILTER STRINGS IN A  
FILTER GRAPH**

Inventor(s):

**Daniel J. Miller**

**Eric H. Rudolph**

ATTORNEY'S DOCKET NO. MS1-642US

1 **TECHNICAL FIELD**

2  
3 This invention generally relates to processing media content and, more  
4 particularly, to an interface and related methods for dynamically generating a filter  
5 graph in a development system.

6 **BACKGROUND**

7  
8 Recent advances in computing power and related technology have fostered  
9 the development of a new generation of powerful software applications. Gaming  
10 applications, communications applications, and multimedia applications have  
11 particularly benefited from increased processing power and clocking speeds.  
12 Indeed, once the province of dedicated, specialty workstations, many personal  
13 computing systems now have the capacity to receive, process and render  
14 multimedia objects (e.g., audio and video content). While the ability to display  
15 (receive, process and render) multimedia content has been around for a while, the  
16 ability for a standard computing system to support true multimedia editing  
17 applications is relatively new.

18 In an effort to satisfy this need, Microsoft Corporation introduced an  
19 innovative development system supporting advanced user-defined multimedia  
20 editing functions. An example of this architecture is presented in US Patent No.  
21 5,913, 038 issued to Griffiths and commonly owned by the assignee of the present  
22 invention, the disclosure of which is expressly incorporated herein by reference.

23 In the '038 patent, Griffiths introduced the an application program interface  
24 which, when exposed to higher-level development applications, enable a user to  
25 graphically construct a multimedia processing project by piecing together a

1 collection of “filters” exposed by the interface. The interface described therein is  
2 referred to as a filter graph manager. The filter graph manager controls the data  
3 structure of the filter graph and the way data moves through the filter graph. The  
4 filter graph manager provides a set of component object model (COM) interfaces  
5 for communication between a filter graph and its application. Filters of a filter  
6 graph architecture are preferably implemented as COM objects, each  
7 implementing one or more interfaces, each of which contains a predefined set of  
8 functions, called methods. Methods are called by an application program or other  
9 component objects in order to communicate with the object exposing the interface.  
10 The application program can also call methods or interfaces exposed by the filter  
11 graph manager object.

12 Filter graphs work with data representing a variety of media (or non-media)  
13 data types, each type characterized by a data stream that is processed by the filter  
14 components comprising the filter graph. A filter positioned closer to the source of  
15 the data is referred to as an upstream filter, while those further down the  
16 processing chain is referred to as a downstream filter. For each data stream that  
17 the filter handles it exposes at least one virtual pin (i.e., distinguished from a  
18 physical pin such as one might find on an integrated circuit). A virtual pin can be  
19 implemented as a COM object that represents a point of connection for a  
20 unidirectional data stream on a filter. Input pins represent inputs and accept data  
21 into the filter, while output pins represent outputs and provide data to other filters.  
22 Each of the filters include at least one memory buffer, wherein communication of  
23 the media stream between filters is accomplished by a series of “copy” operations  
24 from one filter to another.  
25

As introduced in Griffiths, a filter graph has three different types of filters: source filters, transform filters, and rendering filters. A source filter is used to load data from some source; a transform filter processes and passes data; and a rendering filter renders data to a hardware device or other locations (e.g., saved to a file, etc.). An example of a filter graph for a simplistic media rendering process is presented with reference to Fig. 1.

Fig. 1 graphically illustrates an example filter graph for rendering media content. As shown, the filter graph 100 is comprised of a plurality of filters 102-114, which read, process (transform) and render media content from a selected source file. As shown, the filter graph includes each of the types of filters described above, interconnected in a linear fashion.

Products utilizing the filter graph have been well received in the market as it has opened the door to multimedia editing using otherwise standard computing systems. It is to be appreciated, however, that the construction and implementation of the filter graphs are computationally intensive and expensive in terms of memory usage. Even the most simple of filter graphs requires and abundance of memory to facilitate the copy operations required to move data between filters. Thus, complex filter graphs can become unwieldy, due in part to the linear nature of conventional development system architecture. Moreover, it is to be appreciated that the filter graphs themselves consume memory resources, thereby compounding the issue introduced above.

Thus, what is required is a filter graph architecture that reduces the computational and memory resources required to support even the most complex of multimedia projects. Indeed, what is required is a development interface and related methods that dynamically generates a filter graph during project execution,

1 thereby improving the perceived performance of the development system. Just  
2 such a solution is disclosed below.

### 3 **SUMMARY**

4  
5 This invention concerns a system and related interfaces supporting the  
6 processing of media content. In accordance with one aspect of the present  
7 embodiment, a method for processing a development project is presented  
8 comprising generating a source chain for use in a development project, and  
9 caching the source chain when it is not currently required in the development  
10 project. As execution of the development project continues, or during a  
11 subsequent project, if the source processing chain is required, it is retrieved from  
12 cache, modified as necessary to meet the needs of the development project, and  
13 integrated into the development project. It will be appreciated, from the  
14 description to follow, that use of the processing chain caching techniques  
15 described herein provide improved performance characteristics over conventional  
16 development systems.

### 17 **BRIEF DESCRIPTION OF THE DRAWINGS**

18  
19 The same reference numbers are used throughout the figures to reference  
20 like components and features.

21 Fig. 1 is a graphical representation of a conventional filter graph  
22 representing a user-defined development project.

23 Fig. 2 is a block diagram of a computing system incorporating the teachings  
24 of the described embodiment.

1 Fig. 3 is a block diagram of an example software architecture incorporating  
2 the teachings of the described embodiment.

3 Fig. 4 is a graphical illustration of an example software-enabled matrix  
4 switch, according to an exemplary embodiment.

5 Fig. 5 is a graphical representation of a data structure comprising a  
6 programming grid to selectively couple one or more of a scalable plurality of input  
7 pins to a scalable plurality of output pins of the matrix switch filter, in accordance  
8 with one aspect of the described embodiment.

9 Fig. 6 is a graphical illustration denoting shared buffer memory between  
10 filters, according to one aspect of the described embodiment.

11 Fig. 7 is a flow chart of an example method for generating a filter graph, in  
12 accordance with one aspect of the described embodiment.

13 Fig. 8 is a flow chart of an example method for negotiating buffer  
14 requirements between at least two adjacent filters, according to one aspect of the  
15 described embodiment.

16 Fig. 9 graphically illustrates an overview of a process that takes a user-  
17 defined editing project and composites a data structure that can be used to program  
18 the matrix switch.

19 Fig. 10 graphically illustrates the project of Fig. 9 in greater detail.

20 Fig. 11 shows an exemplary matrix switch dynamically generated in  
21 support of the project developed in Figs. 9 and 10, according to one described  
22 embodiment.

23 Fig. 12 illustrates a graphic representation of an exemplary data structure  
24 that represents the project of Fig. 10, according to one described embodiment.  
25

1 Figs. 13-18 graphically illustrate various states of a matrix switch  
2 programming grid at select points in processing the project of Figs. 9 and 10  
3 through the matrix switch, in accordance with one described embodiment.

4 Fig. 19 is a flow chart of an example method for processing media content,  
5 in accordance with one described embodiment.

6 Fig. 20 illustrates an example project with a transition and an effect, in  
7 accordance with one described embodiment.

8 Fig. 21 shows an exemplary data structure in the form of a hierarchical tree  
9 that represents the project of Fig. 20.

10 Figs. 22 and 23 graphically illustrate an example matrix switch  
11 programming grid associated with the project of Fig. 20 at select points in time,  
12 according to one described embodiment.

13 Fig. 24 shows an example matrix switch dynamically generated and  
14 configured as the grid of Figs. 22 and 23 was being processed, in accordance with  
15 one described embodiment.

16 Fig. 25 shows an exemplary project in accordance with one described  
17 embodiment.

18 Fig. 26 graphically illustrates an example audio editing project, according  
19 to one described embodiment.

20 Fig. 27 depicts an example matrix switch programming grid associated with  
21 the project of Fig. 26.

22 Fig. 28 shows an example matrix switch dynamically generated and  
23 configured in accordance with the programming grid of Fig. 27 to perform the  
24 project of Fig. 26, according to one described embodiment.  
25

1 Fig. 29 illustrates an exemplary media processing project incorporating  
2 another media processing project as a composite, according to yet another  
3 described embodiment.

4 Fig. 30 graphically illustrates an example data structure in the form of a  
5 hierarchical tree structure that represents the project of Fig. 29.

6 Figs 31-36 graphically illustrate various matrix switch programming grid  
7 states at select points in generating and configuring the matrix switch to  
8 implement the media processing of Fig. 29.

9 Fig. 38 illustrates an example matrix switch suitable for use in the media  
10 processing project of Fig. 29, according to one described embodiment.

11 Fig. 38a graphically illustrates an example data structure in the form of a  
12 hierarchical tree structure that represents a project that is useful in understanding  
13 composites in accordance with the described embodiments.

14 Fig. 39 is a flow diagram that describes steps in a method in accordance  
15 with one described embodiment.

16 Fig. 40 is a flow chart of an example method for processing media content,  
17 in accordance with another embodiment of the present invention.

18 Fig. 41 is a flow chart of an example method for dynamically generating a  
19 filter graph during execution of a development project, according to one aspect of  
20 the present invention.

21 Fig. 42 illustrates an example data structure utilized to manage dynamic  
22 graph building, according to one embodiment.

23 Fig. 43 graphically illustrates a filter graph during dynamic graph building,  
24 according to one example implementation.



Fig. 44 graphically illustrates a filter graph with thread dependencies during dynamic graph building, according to one embodiment of the present invention.

Fig. 45 is a block diagram of an example filter graph manager incorporating storage space for connected filter strings, in accordance with one aspect of the present invention.

Fig. 46 is a graphical illustration of an example filter cache suitable for use in accordance with the teachings of the present invention.

Fig. 47 is a flow chart of an example method for dynamically loading source filter strings, according to one embodiment of the present invention.

Fig. 48 is a flow chart of an example method for dynamically unloading source filter strings, according to one embodiment of the present invention.

## **DETAILED DESCRIPTION**

### **Related Applications**

This application is related to the following commonly-filed U.S. Patent Applications, all of which are commonly assigned to Microsoft Corp., the disclosures of which are incorporated by reference herein:

- Application Serial No. \_\_\_\_\_, entitled “An Interface and Related Methods for Reducing Source Accesses in a Development System”, naming Daniel J. Miller and Eric H. Rudolph as inventors, and bearing attorney docket number MS1-643US;
- Application Serial No. \_\_\_\_\_, entitled “A System and Related Interfaces Facilitating the Processing of Media Content”, naming Daniel J. Miller and Eric H. Rudolph as inventors, and bearing attorney docket number MS1-629US;
- Application Serial No. \_\_\_\_\_, entitled “A System and Related Methods for Reducing Source Filter Invocation in a Development Project”, naming Daniel J. Miller and Eric H. Rudolph as inventors, and bearing attorney docket number MS1-631US;

- Application Serial No. \_\_\_\_\_, entitled “A System and Related Methods for Reducing Memory Requirements of a Media Processing System”, naming Daniel J. Miller and Eric H. Rudolph as inventors, and bearing attorney docket number MS1-632US;
- Application Serial No. \_\_\_\_\_, entitled “A System and Related Methods for Reducing the Instances of Source Files in a Filter Graph”, naming Daniel J. Miller and Eric H. Rudolph as inventors, and bearing attorney docket number MS1-633US;
- Application Serial No. \_\_\_\_\_, entitled “An Interface and Related Methods for Dynamically Generating a Filter Graph in a Development System”, naming Daniel J. Miller and Eric H. Rudolph as inventors, and bearing attorney docket number MS1-634US;
- Application Serial No. \_\_\_\_\_, entitled “A System and Related Methods for Processing Audio Content in a Filter Graph”, naming Daniel J. Miller and Eric H. Rudolph as inventors, and bearing attorney docket number MS1-639US;
- Application Serial No. \_\_\_\_\_, entitled “Methods and Systems for Processing Media Content”, naming Daniel J. Miller and Eric H. Rudolph as inventors, and bearing attorney docket number MS1-640US;
- Application Serial No. \_\_\_\_\_, entitled “Systems for Managing Multiple Inputs and Methods and Systems for Processing Media Content ”, naming Daniel J. Miller and Eric H. Rudolph as inventors, and bearing attorney docket number MS1-635US;
- Application Serial No. \_\_\_\_\_, entitled “Methods and Systems for Implementing Dynamic Properties on Objects that Support Only Static Properties”, naming Daniel J. Miller and David Maymudes as inventors, and bearing attorney docket number MS1-638US;
- Application Serial No. \_\_\_\_\_, entitled “Methods and Systems for Efficiently Processing Compressed and Uncompressed Media Content”, naming Daniel J. Miller and Eric H. Rudolph as inventors, and bearing attorney docket number MS1-630US;
- Application Serial No. \_\_\_\_\_, entitled “Methods and Systems for Effecting Video Transitions Represented By Bitmaps”, naming Daniel J. Miller and David Maymudes as inventors, and bearing attorney docket number MS1-637US;
- Application Serial No. \_\_\_\_\_, entitled “Methods and Systems for Mixing Digital Audio Signals”, naming Eric H. Rudolph as inventor, and bearing attorney docket number MS1-636US; and
- Application Serial No. \_\_\_\_\_, entitled “Methods and Systems for Processing Multi-media Editing Projects”, naming Eric H. Rudolph as inventor, and bearing attorney docket number MS1-641US.

1 Various described embodiments concern an application program interface  
2 associated with a development system. According to one example  
3 implementation, the interface is exposed to a media processing application to  
4 enable a user to dynamically generate complex media processing tasks, e.g.,  
5 editing projects. In the discussion herein, aspects of the invention are developed  
6 within the general context of computer-executable instructions, such as program  
7 modules, being executed by one or more conventional computers. Generally,  
8 program modules include routines, programs, objects, components, data structures,  
9 etc. that perform particular tasks or implement particular abstract data types.  
10 Moreover, those skilled in the art will appreciate that the invention may be  
11 practiced with other computer system configurations, including hand-held devices,  
12 personal digital assistants, multiprocessor systems, microprocessor-based or  
13 programmable consumer electronics, network PCs, minicomputers, mainframe  
14 computers, and the like. In a distributed computer environment, program modules  
15 may be located in both local and remote memory storage devices. It is noted,  
16 however, that modification to the architecture and methods described herein may  
17 well be made without deviating from spirit and scope of the present invention.  
18 Moreover, although developed within the context of a media processing system  
19 paradigm, those skilled in the art will appreciate, from the discussion to follow,  
20 that the application program interface may well be applied to other development  
21 system implementations. Thus, the media processing system described below is  
22 but one illustrative implementation of a broader inventive concept.  
23  
24  
25

## **Example System Architecture**

**Fig. 2** illustrates an example of a suitable computing environment 200 on which the system and related methods for processing media content may be implemented.

It is to be appreciated that computing environment 200 is only one example of a suitable computing environment and is not intended to suggest any limitation as to the scope of use or functionality of the media processing system. Neither should the computing environment 200 be interpreted as having any dependency or requirement relating to any one or combination of components illustrated in the exemplary computing environment 200.

The media processing system is operational with numerous other general purpose or special purpose computing system environments or configurations. Examples of well known computing systems, environments, and/or configurations that may be suitable for use with the media processing system include, but are not limited to, personal computers, server computers, thin clients, thick clients, handheld or laptop devices, multiprocessor systems, microprocessor-based systems, set top boxes, programmable consumer electronics, network PCs, minicomputers, mainframe computers, distributed computing environments that include any of the above systems or devices, and the like.

In certain implementations, the system and related methods for processing media content may well be described in the general context of computer-executable instructions, such as program modules, being executed by a computer. Generally, program modules include routines, programs, objects, components, data structures, etc. that perform particular tasks or implement particular abstract

1 data types. The media processing system may also be practiced in distributed  
2 computing environments where tasks are performed by remote processing devices  
3 that are linked through a communications network. In a distributed computing  
4 environment, program modules may be located in both local and remote computer  
5 storage media including memory storage devices.

6 In accordance with the illustrated example embodiment of Fig. 2 computing  
7 system 200 is shown comprising one or more processors or processing units 202, a  
8 system memory 204, and a bus 206 that couples various system components  
9 including the system memory 204 to the processor 202.

10 Bus 206 is intended to represent one or more of any of several types of bus  
11 structures, including a memory bus or memory controller, a peripheral bus, an  
12 accelerated graphics port, and a processor or local bus using any of a variety of  
13 bus architectures. By way of example, and not limitation, such architectures  
14 include Industry Standard Architecture (ISA) bus, Micro Channel Architecture  
15 (MCA) bus, Enhanced ISA (EISA) bus, Video Electronics Standards Association  
16 (VESA) local bus, and Peripheral Component Interconnects (PCI) buss also  
17 known as Mezzanine bus.

18 Computer 200 typically includes a variety of computer readable media.  
19 Such media may be any available media that is locally and/or remotely accessible  
20 by computer 200, and it includes both volatile and non-volatile media, removable  
21 and non-removable media.

22 In Fig. 2, the system memory 204 includes computer readable media in the  
23 form of volatile, such as random access memory (RAM) 210, and/or non-volatile  
24 memory, such as read only memory (ROM) 208. A basic input/output system  
25 (BIOS) 212, containing the basic routines that help to transfer information

1 between elements within computer 200, such as during start-up, is stored in ROM  
2 208. RAM 210 typically contains data and/or program modules that are  
3 immediately accessible to and/or presently be operated on by processing unit(s)  
4 202.

5 Computer 200 may further include other removable/non-removable,  
6 volatile/non-volatile computer storage media. By way of example only, Fig. 2  
7 illustrates a hard disk drive 228 for reading from and writing to a non-removable,  
8 non-volatile magnetic media (not shown and typically called a "hard drive"), a  
9 magnetic disk drive 230 for reading from and writing to a removable, non-volatile  
10 magnetic disk 232 (e.g., a "floppy disk"), and an optical disk drive 234 for reading  
11 from or writing to a removable, non-volatile optical disk 236 such as a CD-ROM,  
12 DVD-ROM or other optical media. The hard disk drive 228, magnetic disk drive  
13 230, and optical disk drive 234 are each connected to bus 206 by one or more  
14 interfaces 226.

15 The drives and their associated computer-readable media provide  
16 nonvolatile storage of computer readable instructions, data structures, program  
17 modules, and other data for computer 200. Although the exemplary environment  
18 described herein employs a hard disk 228, a removable magnetic disk 232 and a  
19 removable optical disk 236, it should be appreciated by those skilled in the art that  
20 other types of computer readable media which can store data that is accessible by a  
21 computer, such as magnetic cassettes, flash memory cards, digital video disks,  
22 random access memories (RAMs), read only memories (ROM), and the like, may  
23 also be used in the exemplary operating environment.

24 A number of program modules may be stored on the hard disk 228,  
25 magnetic disk 232, optical disk 236, ROM 208, or RAM 210, including, by way of

1 example, and not limitation, an operating system 214, one or more application  
2 programs 216 (e.g., multimedia application program 224), other program modules  
3 218, and program data 220. In accordance with the illustrated example  
4 embodiment of Fig. 2, operating system 214 includes an application program  
5 interface embodied as a render engine 222. As will be developed more fully  
6 below, render engine 222 is exposed to higher-level applications (e.g., 216) to  
7 automatically assemble filter graphs in support of user-defined development  
8 projects, e.g., media processing projects. Unlike conventional media processing  
9 systems, however, render engine 222 utilizes a scalable, dynamically  
10 reconfigurable matrix switch to reduce filter graph complexity, thereby reducing  
11 the computational and memory resources required to complete a development  
12 project. Various aspects of the innovative media processing system represented by  
13 a computer 200 implementing the innovative render engine 222 will be developed  
14 further, below.

15 Continuing with Fig. 2, a user may enter commands and information into  
16 computer 200 through input devices such as keyboard 238 and pointing device 240  
17 (such as a "mouse"). Other input devices may include a audio/video input  
18 device(s) 253, a microphone, joystick, game pad, satellite dish, serial port, scanner,  
19 or the like (not shown). These and other input devices are connected to the  
20 processing unit(s) 202 through input interface(s) 242 that is coupled to bus 206,  
21 but may be connected by other interface and bus structures, such as a parallel port,  
22 game port, or a universal serial bus (USB).

23 A monitor 256 or other type of display device is also connected to bus 206  
24 via an interface, such as a video adapter 244. In addition to the monitor, personal  
25 computers typically include other peripheral output devices (not shown), such as

1 speakers and printers, which may be connected through output peripheral interface  
2 246.

3 Computer 200 may operate in a networked environment using logical  
4 connections to one or more remote computers, such as a remote computer 250.  
5 Remote computer 250 may include many or all of the elements and features  
6 described herein relative to computer 200 including, for example, render engine  
7 222 and one or more development applications 216 utilizing the resources of  
8 render engine 222.

9 As shown in Fig. 2, computing system 200 is communicatively coupled to  
10 remote devices (e.g., remote computer 250) through a local area network (LAN)  
11 251 and a general wide area network (WAN) 252. Such networking environments  
12 are commonplace in offices, enterprise-wide computer networks, intranets, and the  
13 Internet.

14 When used in a LAN networking environment, the computer 200 is  
15 connected to LAN 251 through a suitable network interface or adapter 248. When  
16 used in a WAN networking environment, the computer 200 typically includes a  
17 modem 254 or other means for establishing communications over the WAN 252.  
18 The modem 254, which may be internal or external, may be connected to the  
19 system bus 206 via the user input interface 242, or other appropriate mechanism.

20 In a networked environment, program modules depicted relative to the  
21 personal computer 200, or portions thereof, may be stored in a remote memory  
22 storage device. By way of example, and not limitation, Fig. 2 illustrates remote  
23 application programs 216 as residing on a memory device of remote computer  
24 250. It will be appreciated that the network connections shown and described are  
25



1 exemplary and other means of establishing a communications link between the  
2 computers may be used.

3 Turning next to **Fig. 3**, a block diagram of an example development system  
4 architecture is presented, in accordance with one embodiment of the present  
5 invention. In accordance with the illustrated example embodiment of Fig. 3,  
6 development system 300 is shown comprising one or more application program(s)  
7 216 coupled to render engine 222 via an appropriate communications interface  
8 302. As used herein, application program(s) 216 are intended to represent any of a  
9 wide variety of applications which may benefit from use of render engine 222  
10 such as, for example a media processing application 224.

11 The communications interface 302 is intended to represent any of a number  
12 of alternate interfaces used by operating systems to expose application program  
13 interface(s) to applications. According to one example implementation, interface  
14 302 is a component object model (COM) interface, as used by operating systems  
15 offered by Microsoft Corporation. As introduced above, COM interface 302  
16 provides a means by which the features of the render engine 222, to be described  
17 more fully below, are exposed to an application program 216.

18 In accordance with the illustrated example implementation of Fig. 3, render  
19 engine 222 is presented comprising source filter(s) 304A-N, transform filter(s)  
20 306A-N and render filter 310, coupled together utilizing virtual pins to facilitate a  
21 user-defined media processing project. According to one implementation, the  
22 filters of system 300 are similar to the filters exposed in conventional media  
23 processing systems. According to one implementation, however, filters are not  
24 coupled via such interface pins. Rather, alternate implementations are envisioned  
25 wherein individual filters (implemented as objects) make calls to other objects,

1 under the control of the render engine 222, for the desired input. Unlike  
2 conventional systems, however, render engine 222 exposes a scalable, dynamically  
3 reconfigurable matrix switch filter 308, automatically generated and dynamically  
4 configured by render engine 222 to reduce the computational and memory  
5 resource requirements often associated with development projects. As introduced  
6 above, the pins (input and/or output) are application interface(s) designed to  
7 communicatively couple other objects (e.g., filters).

8 In accordance with the example implementation of a media processing  
9 system, an application communicates with an instance of render engine 222 when  
10 the application 216 wants to process streaming media content. Render engine 222  
11 selectively invokes and controls an instance of filter graph manager (not shown) to  
12 automatically create a filter graph by invoking the appropriate filters (e.g., source,  
13 transform and rendering). As introduced above, the communication of media  
14 content between filters is achieved by either (1) coupling virtual output pins of one  
15 filter to the virtual input pins of requesting filter; or (2) by scheduling object calls  
16 between appropriate filters to communicate the requested information. As shown,  
17 source filter 304 receives streaming data from the invoking application or an  
18 external source (not shown). It is to be appreciated that the streaming data can be  
19 obtained from a file on a disk, a network, a satellite feed, an Internet server, a  
20 video cassette recorder, or other source of media content. As introduced above,  
21 transform filter(s) 306 take the media content and processes it in some manner,  
22 before passing it along to render filter 310. As used herein, transform filter(s) 306  
23 are intended to represent a wide variety of processing methods or applications that  
24 can be performed on media content. In this regard, transform filter(s) 306 may  
25 well include a splitter, a decoder, a sizing filter, a transition filter, an effects filter,

1 and the like. The function of each of these filters is described more fully in the  
2 Griffiths application, introduced above, and generally incorporated herein by  
3 reference. The transition filter, as used herein, is utilized by render engine 222 to  
4 transition the rendered output from a first source to a second source. The effect  
5 filter is selectively invoked to introduce a particular effect (e.g., fade, wipe, audio  
6 distortion, etc.) to a media stream.

7 In accordance with one aspect of the embodiment, to be described more  
8 fully below, matrix switch filter 308 selectively passes media content from one or  
9 more of a scalable plurality of input(s) to a scalable plurality of output(s).  
10 Moreover, matrix switch 308 also supports implementation of a cascaded  
11 architecture utilizing feedback paths, i.e., wherein transform filters 306B, 306C,  
12 etc. coupled to the output of matrix switch 308 are dynamically coupled to one or  
13 more of the scalable plurality of matrix switch input(s). An example of this  
14 cascaded filter graph architecture is introduced in Fig. 3, and further explained in  
15 example implementations, below.

16 Typically, media processed through source, transform and matrix switch  
17 filters are ultimately passed to render filter 310, which provides the necessary  
18 interface to a hardware device, or other location that accepts the renderer output  
19 format, such as a memory or disk file, or a rendering device.

20 **Fig. 4** is a graphical illustration of an example software-enabled matrix  
21 switch 308, according to one example embodiment of the present invention. As  
22 shown, the matrix switch 308 is comprised of a scalable plurality of input(s) 402  
23 and a scalable plurality of output(s) 404, wherein any one or more of the input(s)  
24 402 may be iteratively coupled to any one or more of the output(s) 404, based on  
25 the content of the matrix switch programming grid 406, automatically generated

1 by render engine 222. According to an alternate implementation introduced  
2 above, switch matrix 308 is programmed by render engine 222 to dynamically  
3 generate object calls to communicate media content between filters. In addition,  
4 according to one implementation, matrix switch 308 includes a plurality of  
5 input/output (I/O) buffers 408, as well as means for maintaining source, or media  
6 time 410 and/or timeline, or project time 412. It is to be appreciated, however,  
7 that in alternate implementations matrix switch 308 does not maintain both source  
8 and project times, relying on an upstream filter to convert between these times. As  
9 will be developed more fully below, matrix switch 308 dynamically couples one or  
10 more of the scalable plurality of inputs 402 to one or more of the scalable plurality  
11 of outputs 404 based, at least in part, on the media time 410 and/or the project time  
12 412 and the content of matrix switch programming grid 406. In this regard, matrix  
13 switch 308 may be characterized as time-aware, supporting such advanced editing  
14 features as searching/seeking to a particular point (e.g., media time) in the media  
15 content, facilitating an innovative buffering process utilizing I/O buffers 408 to  
16 facilitate look-ahead processing of media content, and the like. Thus, it will be  
17 appreciated given the discussion to follow that introduction of the matrix switch  
18 308 provides a user with an editing flexibility that was heretofore unavailable in a  
19 personal computer-based media processing system.

20 As introduced above, the inputs 402 and outputs 404 of matrix switch 308  
21 are interfaces which facilitate the time-sensitive routing of data (e.g., media  
22 content) in accordance with a user-defined development project. Matrix switch  
23 308 has a scalable plurality of inputs 402 and outputs 404, meaning that the  
24 number of inputs 402 and outputs 404 are individually generated to satisfy a given  
25 editing project. Insofar as each of the inputs/outputs (I/O) has an associated

1 transfer buffer (preferably shared with an adjacent filter) to communicate media  
2 content, the scalability of the input/output serves to reduce the overall buffer  
3 memory consumed by an editing project. According to one implementation,  
4 output 1 is generally reserved as a primary output, e.g., coupled to a rendering  
5 filter (not shown).

6 According to one implementation, for each input 402 and output 404,  
7 matrix switch 308 attempts to be the allocator, or manager of the buffer associated  
8 with the I/O(s) shared with adjacent filters. One reason is to ensure that all of the  
9 buffers are of the same size and share common attributes so that a buffer  
10 associated with any input 402 may be shared with any output 404, thereby  
11 reducing the need to copy memory contents between individual buffers associated  
12 with such inputs/outputs. If matrix switch 308 cannot be an allocator for a given  
13 output (404), communication from an input (402) to that output is performed using  
14 a conventional memory copy operation between the individual buffers associated  
15 with the select input/output.

16 As introduced above, the matrix switch programming grid 406 is  
17 dynamically generated by render engine 222 based, at least in part, on the user-  
18 defined development project. As will be developed below, render engine 222  
19 invokes an instance of filter graph manager to assemble a tree structure of an  
20 editing project, noting dependencies between source, filters and time to  
21 dynamically generate the programming grid 406. A data structure comprising an  
22 example programming grid 406 is introduced with reference to Fig. 5, below.

23 Turning briefly to **Fig. 5**, a graphical representation of a data structure  
24 comprising an example programming grid 406 is presented, in accordance with  
25 one embodiment of the present invention. In accordance with the illustrated

1 example embodiment of Fig. 5, programming grid 406 is depicted as a two-  
2 dimensional data structure comprising a column along the y-axis 502 of the grid  
3 denoting input pins associated with a content chain (e.g., series of filters to process  
4 media content) of the development project. The top row along the x-axis 504 of  
5 the data structure denotes project time. With these grid "borders", the body 506 of  
6 the grid 406 is populated with output pin assignments, denoting which input pin is  
7 coupled to which output pin during execution of the development project. In this  
8 way, render engine 222 dynamically generates and facilitates matrix switch 308.  
9 Those skilled in the art will appreciate, however, that data structures of greater or  
10 lesser complexity may well be used in support of the programming grid 406  
11 without deviating from the spirit and scope of the present invention.

12 Returning to Fig. 4, matrix switch 308 is also depicted with a plurality of  
13 input/output buffers 408, shared among all of the input(s)/output(s) (402, 404) to  
14 facilitate advanced processing features. That is, while not required to implement  
15 the core features of matrix switch 308, I/O buffers 408 facilitate a number of  
16 innovative performance enhancing features to improve the performance (or at least  
17 the user's perception of performance) of the processing system, thereby providing  
18 an improved user experience. According to one implementation, I/O buffers 408  
19 are separate from the buffers assigned to each individual input and output pin in  
20 support of communication through the switch. According to one implementation,  
21 I/O buffers 408 are primarily used to foster look-ahead processing of the project.  
22 Assume, for example, that a large portion of the media processing project required  
23 only 50% of the available processing power, while some smaller portion required  
24 150% of the available processing power. Implementation of the shared I/O buffers  
25 408 enable filter graph manager to execute tasks ahead of schedule and buffer this

1 content in the shared I/O buffers 408 until required. Thus, when execution of the  
2 filter graph reaches a point where more than 100% of the available processing  
3 power is required, the processing system can continue to supply content from the  
4 I/O buffers 408, while the system completes execution of the CPU-intensive tasks.  
5 If enough shared buffer space is provided, the user should never know that some  
6 tasks were not performed in real-time. According to one implementation, shared  
7 buffers 408 are dynamically split into two groups by render engine 222, a first  
8 group supports the input(s) 402, while a second (often smaller) group is used in  
9 support of a primary output (e.g., output pin 1) to facilitate a second, independent  
10 output processing thread. The use of an independent output buffers the render  
11 engine from processing delays that might occur in upstream and/or downstream  
12 filters, as discussed above. It will be appreciated by those skilled in the art that  
13 such that matrix switch 308 and the foregoing described architecture beneficially  
14 suited to support media streaming applications.

15 As introduced above, the filter graph is time-aware in the sense that media  
16 (source) time and project execution time are maintained. According to one  
17 implementation, matrix switch 308 maintains at least the project clock, while an  
18 upstream filter maintains the source time, converting between source and project  
19 time for all downstream filters (i.e., including the matrix switch 308). According  
20 to one implementation, the frame rate converter filter of a filter graph is  
21 responsible for converting source time to project time, and vice versa, i.e.,  
22 supporting random seeks, etc. Alternatively, matrix switch 308 utilizes an  
23 integrated set of clock(s) to independently maintain project and media times.

24 Having introduced the architectural and operational elements of matrix  
25 switch filter 308, **Fig. 6** graphically illustrates an example filter graph

1 implementation incorporating the innovative matrix switch 308. In accordance  
2 with the illustrated example embodiment, filter graph 600 is generated by render  
3 engine 222 in response to a user defined development project. Unlike the lengthy  
4 linear filter graphs typical of convention development systems however, filter  
5 graph 600 is shown incorporating a matrix switch filter 308 to recursively route  
6 the pre-processed content (e.g., through filters 602, 606, 610, 614 and 618,  
7 described more fully below) through a user-defined number of transform filters  
8 including, for example, transition filter(s) 620 and effects filter(s) 622. Moreover,  
9 as will be developed more fully below, the scalable nature of matrix switch filter  
10 308 facilitates such iterative processing for any number of content threads, tracks  
11 or compositions.

12 According to one implementation, a matrix switch filter 308 can only  
13 process one type of media content, of the same size and at the same frame-rate  
14 (video) or modulation type/schema (audio). Thus, Fig. 6 is depicted comprising  
15 pre-processing filters with a parser filter 606 to separate, independent content  
16 type(s) (e.g., audio content and video content), wherein one of the media types  
17 would be processed along a different path including a separate instance of matrix  
18 switch 308. Thus, in accordance with the illustrated example embodiment of a  
19 media processing system, processing multimedia content including audio and  
20 video would utilize two (2) matrix switch filters 308, one dedicated to audio  
21 processing (not shown) and one dedicated to video processing. That is not to say,  
22 however, that multiple switch filters 308 could not be used (e.g., two each for  
23 audio and video) for each content type in alternate implementations. Similarly, it  
24 is anticipated that in alternate implementations a matrix switch 308 that accepts  
25



1 multiple media types could well be used without deviating from the spirit and  
2 scope of the present invention.

3 In addition filter graph 600 includes a decoder filter 610 to decode the  
4 media content. Resize filter 614 is employed when matrix switch 308 is to receive  
5 content from multiple sources, ensuring that the size of the received content is the  
6 same, regardless of the source. According to one implementation, resize filter 614  
7 is selectively employed in video processing paths to adjust the media size of  
8 content from one or more sources to a user-defined level. Alternatively, resizer  
9 filter 614 adjusts the media size to the largest size provided by any one or more  
10 media sources. That is, if, for example, render engine 222 identifies the largest  
11 required media size (e.g., 1270x1040 video pixels per frame) and, for any content  
12 source not providing content at this size, the content is modified (e.g., stretched,  
13 packed, etc.) to fill this size requirement. The frame rate converter (FRC) and  
14 pack filter 618, introduced above, ensures that video content from the multiple  
15 sources is arriving at the same frame rate, e.g., ten (10) frames per second. As  
16 introduced above, the FRC also maintains the distinction between source time and  
17 project time.

18 In accordance with one aspect of the present invention, filter graph 600 is  
19 depicted utilizing a single, negotiated buffer 604, 608, 612, 616, etc. between  
20 adjacent filters. In this regard, render engine 222 reduces the buffer memory  
21 requirements in support of a development project.

22 From the point of pre-processing (filters 602, 606, 610, 614, 618), rather  
23 than continue a linear filter graph incorporating all of the transition 620 and effect  
24 622 filter(s), render engine 222 utilizes a cascade architecture, recursively passing  
25 media content through the matrix switch 308 to apply to the transform filter(s)

(e.g., 620, 622, etc.) to complete the execution of the development project. It will be appreciated by those skilled in the art that the ability to recursively pass media content through one or more effect and/or transition filters provided by the matrix switch filter 308 greatly reduces the perceived complexity of otherwise large filter graphs, while reducing memory and computational overhead.

Turning to **Fig. 7**, a flow chart of an example method for generating a filter graph is presented, in accordance with one aspect of the present invention. The method 700 begins with block 702 wherein render engine 222 receives an indication to generate a filter graph representing a user-defined development project (e.g., a media editing project). According to one example implementation, the indication is received from an application 224 via COM interface(s) 302.

In block 704, render engine 222 facilitates generation of the editing project, identifying the number and type of media sources selected by the user. In block 706, based at least in part on the number and/or type of media sources, filter graph manger 222 exposes source, transform and rendering filter(s) to effect a user defined media processing project, while beginning to establish a programming grid 406 for the matrix switch filter 308.

In block 708, reflecting user editing instructions, render engine 222 completes the programming grid 406 for matrix switch 308, identifying which inputs 402 are to be coupled to which outputs 404 at particular project times.

Based, at least in part, on the programming grid 406 render engine 222 generates a matrix switch filter 308 with an appropriate number of input 402 and output 404 pins to effect the project, and assembles the filter graph, block 710.

In block 712, to reduce the buffer memory requirements for the processing project, the render engine 222 instructs the filters populating the filter graph to

1 (re)negotiate buffer memory requirements between filters. That is, adjacent filters  
2 attempt to negotiate a size and attribute standard so that a single buffer can be  
3 utilized to couple each an output pin of one filter to an input pin of a downstream  
4 filter. An example implementation of the buffer negotiation process of block 712  
5 is presented in greater detail with reference to Fig. 8.

6 Turning briefly to Fig. 8, an example method of negotiating buffer  
7 requirements between adjacent filters is presented, in accordance with one  
8 example implementation of the present invention. Once the final connection is  
9 established to matrix switch 308, matrix switch 308 identifies the maximum buffer  
10 requirements for any filter coupled to any of its pins (input 402 and/or output 404),  
11 block 802. According to one implementation, the maximum buffer requirements  
12 are defined as the lowest common multiple of buffer alignment requirements, and  
13 the maximum of all the pre-fix requirements of the filter buffers.

14 In block 804, matrix switch 308 selectively removes one or more existing  
15 filter connections to adjacent filters. Matrix switch 308 then reconnects all of its  
16 pins to adjacent filters using a common buffer size between each of the pins, block  
17 806. In block 808, matrix switch 308 negotiates to be the allocator for all of its  
18 pins (402, 404). If the matrix switch 308 cannot, for whatever reason, be the  
19 allocator for any of its input pins 402 minimal loss to performance is encountered,  
20 as the buffer associated with the input pin will still be compatible with any  
21 downstream filter (i.e., coupled to an output pin) and, thus, the buffer can still be  
22 passed to the downstream filter without requiring a memory copy operation. If,  
23 however, matrix switch 308 cannot be an allocator for one of its output pins 404,  
24 media content must then be transferred to at least the downstream filter associated  
25 with that output pin using a memory copy operation, block 810.

1 In block 812, once the matrix switch 308 has re-established its connection  
2 to adjacent filters, render engine 222 restores the connection in remaining filters  
3 using negotiated buffer requirements emanating from the matrix switch filter 308  
4 buffer negotiations. Once the connections throughout the filter graph have been  
5 reconnected, the process continues with block 714 of Fig. 7.

6 In block 714 (Fig. 7), have re-established the connections between filters,  
7 render engine 222 is ready to implement a user's instruction to execute the media  
8 processing project.

### 9 10 **Example Operation and Implementation(s)**

11 The matrix switch described above is quite useful in that it allows multiple  
12 inputs to be directed to multiple outputs at any one time. These input can compete  
13 for a matrix switch output. The embodiments described below permit these  
14 competing inputs to be organized so that the inputs smoothly flow through the  
15 matrix switch to provide a desired output. And, while the inventive programming  
16 techniques are described in connection with the matrix switch as such is employed  
17 in the context of multi-media editing projects, it should be clearly understood that  
18 application of the inventive programming techniques and structures should not be  
19 so limited only to application in the field of multi-media editing projects or, for  
20 that matter, multi-media applications or data streams. Accordingly, the principles  
21 about to be discussed can be applied to other fields of endeavor in which multiple  
22 inputs can be characterized as competing for a particular output during a common  
23 time period.

24 In the multi-media example below, the primary output of the matrix switch  
25 is a data stream that defines an editing project that has been created by a user.

1 Recall that this editing project can include multiple different sources that are  
2 combined in any number of different ways, and the sources that make up a project  
3 can comprise audio sources, video sources, or both. The organization of the inputs  
4 and outputs of the matrix switch are made manageable, in the examples described  
5 below, by a data structure that permits the matrix switch to be programmed.

6 Fig. 9 shows an overview of a process that takes a user-defined editing  
7 project and renders from it a data structure that can be used to program the matrix  
8 switch.

9 Specifically, a user-defined editing project is shown generally at 900.  
10 Typically, when a user creates an editing project, they can select from a number of  
11 different multimedia clips that they can then assemble into a unique presentation.  
12 Each individual clip represents a *source* of digital data or a source stream (e.g.,  
13 multimedia content). Projects can include one or more sources 902. In defining  
14 their project, a user can operate on sources in different ways. For example, video  
15 sources can have *transitions* 904 and *effects* 906 applied on them. A transition  
16 object is a way to change between two or more sources. As discussed above, a  
17 transition essentially receives as input, two or more streams, operates on them in  
18 some way, and produces a single output stream. An exemplary transition can  
19 comprise, for example, fading from one source to another. An effect object can  
20 operate on a single source or on a composite of sources. An effect essentially  
21 receives a single input stream, operates on it in some way, and produces a single  
22 output stream. An exemplary effect can comprise a black-and-white effect in  
23 which a video stream that is configured for presentation in color format is  
24 rendered into a video stream that is configured for presentation in black and white  
25 format. Unlike conventional effect filters, effect object 906 may well perform

1 multiple effect tasks. That is, in accordance with one implementation, an effect  
2 object (e.g., 906) may actually perform multiple tasks on the received input  
3 stream, wherein said tasks would require multiple effect filters in a conventional  
4 filter graph system.

5 An exemplary user interface 908 is shown and represents what a user might  
6 see when they produce a multimedia project with software executing on a  
7 computer. In this example, the user has selected three sources A, B, and C, and  
8 has assembled the sources into a project timeline. The project timeline defines  
9 when the individual sources are to be rendered, as well as when any transitions  
10 and/or effects are to occur.

11 In the discussion that follows, the notion of a *track* is introduced. A track  
12 can contain one or more sources or source clips. If a track contains more than one  
13 source clip, the source clips cannot overlap. If source clips are to overlap (e.g.  
14 fading from one source to another, or having one source obscure another), then  
15 multiple tracks are used. A track can thus logically represent a layer on which  
16 sequential video is produced. User interface 908 illustrates a project that utilizes  
17 three tracks, each of which contains a different source. In this particular project  
18 source A will show for a period of time. At a defined time in the presentation,  
19 source A is obscured by source B. At some later time, source B transitions to  
20 source C.

21 In accordance with the described embodiment, the user-defined editing  
22 project 900 is translated into a data structure 910 that represents the project. In the  
23 illustrated and described example, this data structure 910 comprises a tree  
24 structure. It is to be understood, however, that other data structures could be used.  
25 The use of tree structures to represent editing projects is well-known and is not

1 described here in any additional detail. Once the data structure 910 is defined, it is  
2 processed to provide a data structure 912 that is utilized to program the matrix  
3 switch. In the illustrated and described embodiment, data structure 912 comprises  
4 a grid from which the matrix switch can be programmed. It is to be understood  
5 and appreciated that other data structures and techniques could, however, be used  
6 to program the matrix switch without departing from the spirit and scope of the  
7 claimed subject matter.

8 The processing that takes place to define data structures 910 and 912 can  
9 take place using any suitable hardware, software, firmware, or combination  
10 thereof. In the examples set forth below, the processing takes place utilizing  
11 software in the form of a video editing software package that is executable on a  
12 general purpose computer.

### 13 Example Project

14 For purposes of explanation, consider Fig. 10 which shows project 908  
15 from Fig. 9 in a little additional detail. Here, a time line containing numbers 0-16  
16 is provided adjacent the project to indicate when particular sources are to be seen  
17 and when transitions and effects (when present) are to occur. In the examples in  
18 this document, the following convention exists with respect to projects, such as  
19 project 908. A priority exists for video portions of the project such that as one  
20 proceeds from top to bottom, the priority increases. Thus, in the Fig. 10 example,  
21 source A has the lowest priority followed by source B and source C. Thus, if there  
22 is an overlap between higher and lower priority sources, the higher priority source  
23 will prevail. For example, source B will obscure source A from between  $t = 4-8$ .  
24  
25

In this example, the following can be ascertained from the project 908 and time line: from time  $t=0-4$  source A should be routed to the matrix switch's primary output; from  $t=4-12$  source B should be routed to the matrix switch's primary output; from  $t=12-14$  there should be a transition between source B and source C which should be routed to the matrix switch's primary output; and from  $t=14-16$  source C should be routed to the matrix switch's primary output. Thus, relative to the matrix switch, each of the sources and the transition can be characterized by where it is to be routed at any given time. Consider, for example, the table just below:

Object	Routing for a given time
C	$t = 0-12$ (nowhere); $t = 12-14$ (transition); $t = 14-16$ (primary output)
B	$t = 0-4$ (nowhere); $t = 4-12$ (primary output); $t = 12-14$ (transition); $t = 14-16$ (nowhere)
A	$t = 0-4$ (primary output); $t = 4-16$ (nowhere)
Transition	$t = 0-12$ (nowhere); $t = 12-14$ (primary output); $t = 14-16$ (nowhere)

Fig. 11 shows an exemplary matrix switch 1100 that can be utilized in the presentation of the user's project. Matrix switch 1100 comprises multiple inputs and multiple outputs. Recall that a characteristic of the matrix switch 1100 is that any of the inputs can be routed to any of the outputs at any given time. A transition element 1102 is provided and represents the transition that is to occur between sources B and C. Notice that the matrix switch includes four inputs numbered 0-3 and three outputs numbered 0-2. Inputs 0-2 correspond respectively to sources A-C, while input 3 corresponds to the output of the transition element



1 1102. Output 0 corresponds to the switch's primary output, while outputs 1 and 2  
2 are routed to the transition element 1102.

3 The information that is contained in the table above is the information that  
4 is utilized to program the matrix switch. The discussion presented below describes  
5 but one implementation in which the information contained in the above table can  
6 be derived from the user's project time line.

7 Recall that as a user edits or creates a project, software that comprises a part  
8 of their editing software builds a data structure that represents the project. In the  
9 Fig. 9 overview, this was data structure 910. In addition to building the data  
10 structure that represents the editing project, the software also builds and configures  
11 a matrix switch that is to be used to define the output stream that embodies the  
12 project. Building and configuring the matrix switch can include building the  
13 appropriate graphs (e.g., a collection of software objects, or filters) that are  
14 associated with each of the sources and associating those graphs with the correct  
15 inputs of the matrix switch. In addition, building and configuring the matrix  
16 switch can also include obtaining and incorporating additional appropriate filters  
17 with the matrix switch, e.g. filters for transitions, effects, and mixing (for audio  
18 streams). This will become more apparent below.

19 Fig. 12 shows a graphic representation of an exemplary data structure 1200  
20 that represents the project of Fig. 10. Here, the data structure comprises a  
21 traditional hierarchical tree structure. Any suitable data structure can, however, be  
22 utilized. The top node 1202 constitutes a *group* node. A *group* encapsulates a type  
23 of media. For example, in the present example the media type comprises video.  
24 Another media type is audio. The group node can have child nodes that are either  
25 tracks or composites. In the present example, three track nodes 1204, 1206, and

1 1208 are shown. Recall that each track can have one or more sources. If a track  
2 comprises more than one source, the sources cannot overlap. Here, all of the  
3 sources (A, B, and C) overlap. Hence, three different tracks are utilized for the  
4 sources. In terms of priority, the lowest priority source is placed into the tree  
5 furthest from the left at 1204a. The other sources are similarly placed. Notice that  
6 source C (1208a) has a transition 1210 associated with it. A transition object, in  
7 this example, defines a two-input/one output operation. When applied to a track  
8 or a composition (discussed below in more detail), the transition object will  
9 operate between the track to which it has been applied, and any objects that are  
10 beneath it in priority and at the same level in the tree. A "tree level" has a  
11 common depth within the tree and belongs to the same parent. Accordingly, in  
12 this example, the transition 1210 will operate on a source to the left of the track on  
13 which source C resides, and beneath it in priority, i.e. source B. If the transition is  
14 applied to any object that has nothing beneath it in the tree, it will transition from  
15 blackness (and/or silence if audio is included).

16 Once a data structure representing the project has been built, in this case a  
17 hierarchical tree structure, a rendering engine processes the data structure to  
18 provide another data structure that is utilized to program the matrix switch. In the  
19 Fig. 9 example, this additional data structure is represented at 912. It will be  
20 appreciated and understood that the nodes of tree 1200 can include so-called meta  
21 information such as a name, ID, and a time value that represents when that  
22 particular node's object desires to be routed to the output, e.g. node 1204a would  
23 include an identifier for the node associating it with source A, as well as a time  
24 value that indicates that source A desires to be routed to the output from time  $t = 0$ -  
25

1 8. This meta information is utilized to build the data structure that is, in turn,  
2 utilized to program the matrix switch.

3 In the example about to be described below, a specific data structure in the  
4 form of a grid is utilized. In addition, certain specifics are described with respect  
5 to how the grid is processed so that the matrix switch can be programmed. It is to  
6 be understood that the specific described approach is for exemplary purposes only  
7 and is not intended to limit application of the claims. Rather, the specific approach  
8 constitutes but one way of implementing broader conceptual notions embodied by  
9 the inventive subject matter.

10 Figs. 13-18 represent a process through which the inventive grid is built. In  
11 the grid about to be described, the x axis represents time, and the y axis represents  
12 layers in terms of priority that go from lowest (at the top of the grid) to highest (at  
13 the bottom of the grid). Every row in the grid represents the video layer.  
14 Additionally, entries made within the grid represent output pins of the matrix  
15 switch. This will become apparent below.

16 The way that the grid is built in this example is that the rendering engine  
17 does a traversal operation on the tree 1200. In this particular example, the  
18 traversal operation is known as a “depth-first, left-to-right” traversal. This  
19 operation will layerize the nodes so that the leftmost track or source has the lowest  
20 priority and so on. Doing the above-mentioned traversal on tree 1200 (Fig. 12),  
21 the first node encountered is node 1204 which is associated with source A. This is  
22 the lowest priority track or source. A first row is defined for the grid and is  
23 associated with source A. After the first grid row is defined, a grid entry is made  
24 and represents the time period for which source A desires to be routed to the  
25 matrix switch’s primary output.

Fig. 13 shows the state of a grid 1300 after this first processing step. Notice that from time  $t = 0-8$ , a “0” has been placed in the grid. The “0” represents the output pin of the matrix switch—in this case the primary output. Next, the traversal encounters node 1206 (Fig. 12) which is associated with source B. A second row is thus defined for the grid and is associated with source B. After the second grid row is defined, a grid entry is made and represents the time period for which source B desires to be routed to the matrix switch’s primary output.

Fig. 14 shows the state of grid 1300 after this second processing step. Notice that from time  $t = 4-14$ , a “0” has been placed in the grid. Notice at this point that something interesting has occurred which will be resolved below. Each of the layers has a common period of time (i.e.  $t = 4-8$ ) for which it desires to be routed to the matrix switch’s primary output. However, because of the nature of the matrix switch, only one input can be routed to the primary output at a time. Next, the traversal encounters node 1208 (Fig. 12) which is associated with source C. In this particular processing example, a rule is defined that sources on tracks are processed before transitions on the tracks are processed because transitions operate on two objects that are beneath them. A third row is thus defined for the grid and is associated with source C. After the third row is defined, a grid entry is made and represents the time period for which source C desires to be routed to the matrix switch’s primary output.

Fig. 15 shows the state of grid 1300 after this third processing step. Notice that from time  $t = 12-16$ , a “0” has been placed in the grid. Next, the traversal encounters node 1210 (Fig. 12) which corresponds to the transition. Thus, a fourth row is defined in the grid and is associated with the transition. After the fourth

row is defined, a grid entry is made and represents the time period for which the transition desires to be routed to the matrix switch's primary output.

Fig. 16 shows the state of grid 1300 after this fourth processing step. Notice that from time  $t = 12-14$ , a "0" has been placed in the grid for the transition entry. The transition is a special grid entry. Recall that the transition is programmed to operate on two inputs and provide a single output. Accordingly, starting at the transition entry in the grid and working backward, each of the entries corresponding to the same tree level are examined to ascertain whether they contain entries that indicate that they want to be routed to the output during the same time that the transition is to be routed to the output. If grid entries are found that conflict with the transition's grid entry, the conflicting grid entry is changed to a value that corresponds to an output pin that serves as an input to the transition element 1102 (Fig. 11). This is essentially a redirection operation. In the illustrated grid example, the transition first finds the level that corresponds to source C. This level conflicts with the transition's grid entry for the time period  $t = 12-14$ . Thus, for this time period, the grid entry for level C is changed to a switch output that corresponds to an input for the transition element. In this example, a "2" is placed in the grid to signify that for this given time period, this input is routed to output pin 2. Similarly, continuing up the grid, the next level that conflicts with the transition's grid entry is the level that corresponds to source B. Thus, for the conflicting time period, the grid entry for level B is changed to a switch output that corresponds to an input for the transition element. In this example, a "1" is placed in the grid to signify that for this given time period, this input is routed to output pin 1 of the matrix switch.

Fig. 17 shows the state of the grid at this point in the processing. Next, a pruning function is implemented which removes any other lower priority entry that is contending for the output with a higher priority entry. In the example, the portion of A from  $t=4-8$  gets removed because the higher priority B wants the output for that time.

Fig. 18 shows the grid with a cross-hatched area that signifies that portion of A's grid entry that has been removed.

At this point, the grid is in a state in which it can be used to program the matrix switch. The left side entries -- A, B, C, and TRANS represent input pin numbers 0, 1, 2, and 3 (as shown) respectively, on the matrix switch shown in Fig. 11. The output pin numbers of the matrix switch are designated at 0, 1, and 2 both on the switch in Fig. 11 and within the grid in Fig. 18. As one proceeds through the grid, starting with source A, the programming of the matrix switch can be ascertained as follows: A is routed to output pin 0 of the matrix switch (the primary output) from  $t = 0-4$ . From  $t = 4-16$ , A is not routed to any output pins. From  $t = 0-4$ , B is not routed to any of the output pins of the matrix switch. From  $t = 4-12$ , B is routed to the primary output pin 0 of the matrix switch. From  $t = 12-14$ , B is routed to output pin 1 of the matrix switch. Output pin 1 of the matrix switch corresponds to one of the input pins for the transition element 1102 (Fig. 11). From  $t = 14-16$ , B is not routed to any of the output pins of the matrix switch. From  $t = 0-12$ , C is not routed to any of the output pins of the matrix switch. From  $t = 12-14$ , C is routed to output pin 2 of the matrix switch. Output pin 2 of the matrix switch corresponds to one of the input pins for the transition element 302 (Fig. 3). From  $t = 12-14$  the transition element (input pin 3) is routed to output pin 0. From  $t = 14-16$ , C is routed to output pin 0 of the matrix switch.

1 As alluded to above, one of the innovative aspects of the matrix switch 308  
2 is its ability to seek to any point in a source, without having to process the  
3 intervening content serially through the filter. Rather, matrix switch 308 identifies  
4 an appropriate transition point and dumps at least a subset of the intervening  
5 content, and continues processing from the sought point in the content.

6 The ability of the matrix switch 308 to seek to any point in the media  
7 content gives rise to certain performance enhancement heretofore unavailable in  
8 computer implemented media processing systems. For example, generation of a  
9 filter graph by render engine 222 may take into account certain performance  
10 characteristics of the media processing system which will execute the user-defined  
11 media processing project. In accordance with this example implementation,  
12 render engine 222 may access and analyze the system registry of the operating  
13 system, for example, to ascertain the performance characteristics of hardware  
14 and/or software elements of the computing system implementing the media  
15 processing system, and adjust the filter graph construction to improve the  
16 perceived performance of the media processing system by the user. Nonetheless,  
17 there will always be a chance that a particular instance of a filter graph will not be  
18 able to process the media stream fast enough to provide the desired output at the  
19 desired time, i.e., processing of the media stream bogs down leading to delays at  
20 the rendering filter. In such a case, matrix switch 308 will recognize that it is not  
21 receiving media content at the appropriate project time, and may skip certain  
22 sections of the project in an effort to "catch-up" and continue the remainder of the  
23 project in real time. According to one implementation, when matrix switch 308  
24 detects such a lag in processing, it will analyze the degree of the lag and issue a  
25 seek command to the source (through the source processing chain) to a future

1 point in the project, where processing continues without processing any further  
2 content prior to the seeked point.

3 Thus, for the editing project depicted in Fig. 10, the processing described  
4 above first builds a data structure (i.e. data structure 1200 in Fig. 12) that  
5 represents the project in hierarchical space, and then uses this data structure to  
6 define or create another data structure that can be utilized to program the matrix  
7 switch.

8 Fig. 19 is a flow diagram that describes steps in a method in accordance  
9 with the described embodiment. The method can be implemented in any suitable  
10 hardware, software, firmware, or combination thereof. In the illustrated and  
11 described embodiment, the method is implemented in software.

12 Step 1900 provides a matrix switch. An exemplary matrix switch is  
13 described above. Step 1902 defines a first data structure that represents the editing  
14 project. Any suitable data structure can be used, as will be apparent to those of  
15 skill in the art. In the illustrated and described embodiment, the data structure  
16 comprises a hierarchical tree structure having nodes that can represent tracks  
17 (having one or more sources), composites, transitions and effects. Step 1904  
18 processes the first data structure to provide a second data structure that is  
19 configured to program the matrix switch. Any suitable data structure can be  
20 utilized to implement the second data structure. In the illustrated and described  
21 embodiment, a grid structure is utilized. Exemplary processing techniques for  
22 processing the first data structure to provide the second data structure are  
23 described above. Step 1906 then uses the second data structure to program the  
24 matrix switch.



### Example Project with a Transition and an Effect

Consider project 2000 depicted in Fig. 20. In this project there are three tracks, each of which contains a source, i.e. source A, B and C. This project includes an effect applied on source B and a transition between sources B and C. The times are indicated as shown.

As the user creates their project, a data structure representing the project is built. Fig. 21 shows an exemplary data structure in the form of a hierarchical tree 2100 that represents project 2000. There, the data structure includes three tracks, each of which contains one of the sources. The sources are arranged in the tree structure in the order of their priority, starting with the lowest priority source on the left and proceeding to the right. There is an effect (i.e. “Fx”) that is attached to or otherwise associated with source B. Additionally, there is a transition attached to or otherwise associated with source C.

In building the grid for project 2000, the following rule is employed for effects. An effect, in this example, is a one-input/one-output object that is applied to one object—in this case source B. When the effect is inserted into the grid, it looks for any one object beneath it in priority that has a desire to be routed to the primary output of the matrix switch at the same time. When it finds a suitable object, it redirects that object’s output from the matrix switch’s primary output to an output associated with the effect.

As an example, consider Fig. 22 and the grid 2200. At this point in the processing of tree 2100, the rendering engine has incorporated entries in the grid corresponding to sources A, B and the effect. It has done so by traversing the tree 2100 in the above-described way. In this example, the effect has already looked for an object beneath it in priority that is competing for the primary output of the

1 matrix switch. It found an entry for source B and then redirected B's grid entry to  
2 a matrix switch output pin that corresponds to the effect—here output pin 1.

3 As the render engine 222 completes its traversal of tree 2100, it completes  
4 the grid. Fig. 23 shows a completed grid 2200. Processing of the grid after that  
5 which is indicated in Fig. 22 takes place substantially as described above with  
6 respect to the first example. Summarizing, this processing though: after the effect  
7 is entered into the grid and processed as described above, the traversal of tree 2100  
8 next encounters the node associated with source C. Thus, a row is added in the  
9 grid for source C and an entry is made to indicate that source C desires the output  
10 from  $t = 12-16$ . Next, the tree traversal encounters the node associated with the  
11 transition. Accordingly, a row is added to the grid for the transition and a grid  
12 entry is made to indicate that the transition desires the output from  $t = 12-14$ .  
13 Now, as described above, the grid is examined to find two entries, lower in  
14 priority than the transition and located at the same tree level as the transition, that  
15 compete for the primary output of the matrix switch. Here, those entries  
16 correspond to the grid entries for the effect and source C that occur from  $t = 12-14$ .  
17 These grid entries are thus redirected to output pins of the matrix switch 308 that  
18 correspond to the transition—here pins 2 and 3 as indicated. Next, the grid is  
19 pruned which, in this example, removes a portion of the grid entry corresponding  
20 to source A for  $t = 4-8$  because of a conflict with the higher-priority entry for  
21 source B.

22 Fig. 24 shows the resultant matrix switch that has been built and configured  
23 as the grid was being processed above. At this point, the grid can be used to  
24 program the matrix switch. From the grid picture, it is very easy to see how the  
25 matrix switch 308 is going to be programmed. Source A will be routed to the

1 matrix switch's primary output (pin 0) from  $t = 0-4$ ; source B will be redirected to  
2 output pin 1 (effect) from  $t = 4-14$  and the effect on B will be routed to the output  
3 pin 0 from  $t = 4-12$ . From  $t = 12-14$ , the effect and source C will be routed to  
4 output pins corresponding to the transition (pins 2 and 3) and, accordingly, during  
5 this time the transition (input pin 4) will be routed to the primary output (output  
6 pin 0) of the matrix switch. From  $t = 14-16$ , source C will be routed to the primary  
7 output of the matrix switch.

8 It will be appreciated that as the software, in this case the render engine  
9 222, traverses the tree structure that represents a project, it also builds the  
10 appropriate graphs and adds the appropriate filters and graphs to the matrix switch.  
11 Thus, for example, as the render engine 222 encounters a tree node associated with  
12 source A, in addition to adding an entry to the appropriate grid, the software builds  
13 the appropriate graphs (i.e. collection of linked filters), and associates those filters  
14 with an input of the matrix switch. Similarly, when the render engine 222  
15 encounters an effect node in the tree, the software obtains an effect object or filter  
16 and associates it with the appropriate output of the matrix switch. Thus, in the  
17 above examples, traversal of the tree structure representing the project also enables  
18 the software to construct the appropriate graphs and obtain the appropriate objects  
19 and associate those items with the appropriate inputs/outputs of the matrix switch  
20 308. Upon completion of the tree traversal and processing of the grid, an  
21 appropriate matrix switch has been constructed, and the programming (i.e. timing)  
22 of inputs to outputs for the matrix switch has been completed.

### 23 24 Treatment of "blanks" in a Project 25

1        There may be instances in a project when a user leaves a blank in the  
2 project time line. During this blank period, no video or audio is scheduled for  
3 play.

4        Fig. 25 shows a project that has such a blank incorporated therein. If there  
5 is such a blank left in a project, the software is configured to obtain a “black”  
6 source and associate the source with the matrix switch at the appropriate input pin.  
7 The grid is then configured when it is built to route the black source to the output  
8 at the appropriate times and fade from the black (and silent) source to the next  
9 source at the appropriate times. The black source can also be used if there is a  
10 transition placed on a source for which there is no additional source from which to  
11 transition.

### 12        **Audio Mixing**

13        In the examples discussed above, sources comprising video streams were  
14 discussed. In those examples, at any one time, only two video streams were  
15 combined into one video stream. However, each project can, and usually does  
16 contain an audio component. Alternately, a project can contain only an audio  
17 component. The audio component can typically comprise a number of different  
18 audio streams that are combined. The discussion below sets forth but one way of  
19 processing and combining audio streams.  
20

21        In the illustrated example, there is no limit on the number of audio streams  
22 that can be combined at any one time.

23        Suppose, for example, there is an audio project that comprises 5 tracks, A-  
24 E. Fig. 26 shows an exemplary project. The shaded portions of each track  
25 represent the time during which the track is not playing. So, for example, at  $t=0-4$ ,

1 tracks B, D, and E are mixed together and will play. From  $t = 4-10$ , tracks A-E are  
2 mixed together and will play, and the like.

3 Fig. 27 shows the grid for this project at 2700. Since we are dealing with  
4 this composition now, all of the effects and transitions including the audio mixing  
5 are only allowed to affect things in this composition. Thus, there is the concept of  
6 a boundary 2702 that prevents any actions or operations in this composition from  
7 affecting any other grid entries. Note that there are other entries in the grid and  
8 that the presently-illustrated entries represent only those portions of the project  
9 that relate to the audio mixing function.

10 Grid 2700 is essentially set up in a manner similar to that described above  
11 with respect to the video projects. That is, for each track, a row is added to the  
12 grid and a grid entry is made for the time period during which the source on that  
13 track desires to be routed to the primary output of the matrix switch. In the  
14 present example, grid entries are made for sources A-E. Next, in the same way  
15 that a transition or effect was allocated a row in the grid, a "mix" element is  
16 allocated a row in the grid as shown and a grid entry is made to indicate that the  
17 mix element desires to be routed to the primary output of the matrix switch for a  
18 period of time during which two or more sources compete for the matrix switch's  
19 primary output. Note that in this embodiment, allocation of a grid row for the mix  
20 element can be implied. Specifically, whereas in the case of a video project,  
21 overlapping sources simply result in playing the higher priority source (unless the  
22 user defines a transition between them), in the audio realm, overlapping sources  
23 are treated as an implicit request to mix them. Thus, the mix element is allocated a  
24 grid row any time there are two or more overlapping sources.

Once the mix element is allocated into the grid, the grid is processed to redirect any conflicting source entries to matrix switch output pins that correspond to the mix element. In the above case, redirection of the grid entries starts with pin 3 and proceeds through to pin 7. The corresponding matrix switch is shown in Fig. 28. Notice that all of the sources are now redirected through the mix element which is a multi-input/one output element. The mix element's output is fed back around and becomes input pin 15 of the matrix switch. All of the programming of the matrix switch is now reflected in the grid 2700. Specifically, for the indicated time period in the grid, each of the sources is routed to the mix element which, in turn, mixes the appropriate audio streams and presents them to the primary output pin 0 of the matrix switch.

### Compositions

There are situations that can arise when building an editing project where it would be desirable to apply an effect or a transition on just a subset of a particular project or track. Yet, there is no practicable way to incorporate the desired effect or transition. In the past, attempts to provide added flexibility for editing projects have been made in the form of so called "bounce tracks", as will be appreciated and understood by those of skill in the art. The use of bounce tracks essentially involves processing various video layers (i.e. tracks), writing or moving the processed layers or tracks to another location, and retrieving the processed layers when later needed for additional processing with other layers or tracks. This type of processing can be slow and inefficient.

To provide added flexibility and efficiency for multi-media editing projects, the notion of a *composite or composition* is introduced. A composite or

1 composition can be considered as a representation of an editing project as a single  
2 track. Recall that editing projects can have one or more tracks, and each track can  
3 be associated with one or more sources that can have effects applied on them or  
4 transitions between them. In addition, compositions can be nested inside one  
5 another.

### 6 7 Example Project with Composite

8 Consider, for example, Fig. 29 which illustrates an exemplary project 2900  
9 having a composition 2902. In this example, composition 2902 comprises sources  
10 B and C and a transition between B and C that occurs between  $t = 12-14$ . This  
11 composition is treated as an individual track or layer. Project 2900 also includes a  
12 source A, and a transition between source A and composition 2902 at  $t = 4-8$ . It  
13 will be appreciated that compositions can be much more complicated than the  
14 illustrated composition, which is provided for exemplary purposes only.  
15 Compositions are useful because they allow the grouping of a particular set of  
16 operations on one or more tracks. The operation set is performed on the grouping,  
17 and does not affect tracks that are not within the grouping. To draw an analogy, a  
18 composition is similar in principle to a mathematical parenthesis. Those  
19 operations that appear within the parenthesis are carried out in conjunction with  
20 those operations that are intended to operate of the subject matter of the  
21 parenthesis. The operations within the parenthesis do not affect tracks that do not  
22 appear within the parenthesis.

23 In accordance with the processing that is described above in connection  
24 with Fig. 19, a first data structure is defined that represents the editing project.  
25 Fig. 30 shows an exemplary data structure 3000 in the form of a hierarchical tree

1 structure. In this example, group node 3002 includes two children—track node  
2 3004 and composite node 3006. Track node 3004 is associated with source A.  
3 Composite node 3006 includes two children—track nodes 3008 and 3010 that are  
4 respectively associated with sources B (3008a) and C (3010a). A transition T2  
5 (3012) is applied on source C and a transition T1 (3014) is applied on composition  
6 3006.

7 Next, data structure 3000 is processed to provide a second data structure  
8 that is configured to program the matrix switch. Note that as the data structure is  
9 being programmed, a matrix switch is being built and configured at the same time.  
10 In this example, the second data structure comprises a grid structure that is  
11 assembled in much the same way as was described above. There are, however,  
12 some differences and, for purposes of understanding, the complete evolution of the  
13 grid structure is described here. In the discussion that follows, the completed  
14 matrix switch is shown in Fig. 38.

15 When the rendering engine initiates the depth-first, left-to-right traversal of  
16 data structure 3000, the first node it encounters is track node 3004 which is  
17 associated with source A. Thus, a first row of the grid is defined and a grid entry  
18 is made that represents the time period for which source A desires to be routed to  
19 the matrix switch's primary output pin.

20 Fig. 31 shows the state of a grid 3100 after this first processing step. Next  
21 the traversal of data structure 3000 encounters the composite node 3006. The  
22 composite node is associated with two tracks—track 3008 and track 3010. Track  
23 3008 is associated with source B. Accordingly, a second row of the grid is defined  
24 and a grid entry is made that represents the time period for which source B desires  
25 to be routed to the matrix switch's primary output pin. Additionally, since B is a



1 member of a composition, meta-information is contained in the grid that indicates  
2 that this grid row defines one boundary of the composition. This meta-  
3 information is graphically depicted with a bracket that appears to the left of the  
4 grid row.

5 Fig. 32 shows the state of grid 3100 after this processing step. Next, the  
6 traversal of data structure 3000 encounters node 3010 which is associated with  
7 source C. Thus, a third row of the grid is added and a grid entry is made that  
8 represents the time period for which source C desires to be routed to the matrix  
9 switch's primary output pin.

10 Fig. 33 shows the state of grid 3100 after this processing step. Notice that  
11 the bracket designating the composition now encompasses the grid row associated  
12 with source C. The traversal next encounters node 3012 which is the node  
13 associated with the *second* transition T2. Thus, as in the above example, a grid  
14 row is added for the transition and a grid entry is made that represents the time  
15 period for which the transition desires to be routed to the matrix switch's primary  
16 output pin.

17 Fig. 34 shows the state of grid 3100 after this processing step. Notice that  
18 the bracket designating the composition is now completed and encompasses grid  
19 row entries that correspond to sources B and C and the transition between them.  
20 Recall from the examples above that a transition, in this example, is programmed  
21 to operate on two inputs and provide a single output. In this instance, and because  
22 the transition occurs within a composition, the transition is constrained by a rule  
23 that does not allow it to operate on any elements outside of the composition.  
24 Thus, starting at the transition entry and working backward through the grid,  
25 entries at the same tree level and within the composition (as designated by the

1 bracket) are examined to ascertain whether they contain entries that indicate that  
2 they want to be routed to the output during the same time that the transition is to  
3 be routed to the output. Here, both of the entries for sources B and C have  
4 portions that conflict with the transition's entry. Accordingly, those portions of  
5 the grid entries for sources B and C are redirected or changed to correspond to  
6 output pins that are associated with a transition element that corresponds to  
7 transition T2.

8 Fig. 35 shows the state of grid 3100 after this processing step. The  
9 traversal next encounters node 3014 which is the node that is associated with the  
10 transition that occurs between source A and composition 2902 (Fig. 29).  
11 Processing of this transition is similar to processing of the transition immediately  
12 above except for the fact that the transition does not occur within the composition.  
13 Because the transition occurs between the composition and another source, one of  
14 the inputs for the transition will be the composition, and one of the inputs will be  
15 source A (which is outside of the composition). Thus, a grid row is added for this  
16 transition and a grid entry is made that represents the time period for which the  
17 transition desires to be routed to the matrix switch's primary output pin.

18 Fig. 36 shows the state of grid 3100 after this processing step. At this point  
19 then, the grid is examined for entries that conflict with the entry for transition T1.  
20 One conflicting grid entry is found for the row that corresponds to source B (inside  
21 the composition) and one that corresponds to source A (outside the composition).  
22 Accordingly, those portions of the grid row that conflict with transition T1 are  
23 changed or redirected to have values that are associated with output pins of the  
24 matrix switch that are themselves associated with a transition element T1. In this  
25 example, redirection causes an entry of "3" and "4" to be inserted as shown.

Fig. 37 shows the state of grid 3100 after this processing step. If necessary, a pruning operation would further ensure that the grid has no competing entries for the primary output of the matrix switch. The associated input pin numbers of the matrix switch are shown to the left of grid 3100.

Fig. 38 shows a suitably configured matrix switch that has been build in accordance with the processing described above. Recall that, as data structure 3000 (Fig. 30) is processed by the rendering engine, a matrix switch is built and configured in parallel with the building and processing of the grid structure that is utilized to program the matrix switch. From the matrix switch and grid 3100 of Fig. 37, the programming of the switch can be easily ascertained.

Fig. 38a shows an exemplary data structure that represents a project that illustrates the usefulness of composites. In this example, the project can mathematically be represented as follows:

(Fx-noisy (A Tx-Blend B)) Tx-Blend C

Here, an effect (noisy) is applied to A blended with B, the result of which is applied to a blend with C. The composite in this example allows the grouping of the things beneath it so that the effect (noisy), when it is applied, is applied to everything that is beneath it. Notice that without the composite node, there is no node where an effect can be applied that will affect (A Tx-Blend B). Hence, in this example, operations that appear within the parenthesis are carried out on tracks that appear within the parenthesis. Those operations do not affect tracks that are not within the parenthesis.

1 Fig. 39 is a flow diagram that described steps in a method in accordance  
2 with one embodiment. The method can be implemented in any suitable hardware,  
3 software, firmware, or combination thereof. In the presently-described example,  
4 the method is implemented in software.

5 Step 3900 defines a multimedia editing project that includes at least one  
6 composite. The composite represents multiple tracks as a single track for purposes  
7 of the processing described just below. It is important to note that, in the  
8 processing described just below, and because of the use of composites, the extra  
9 processing that is required by bounce tracks is avoided (i.e. operating on two  
10 tracks, moving the operation result to another location, and retrieving the  
11 operation result when later needed). This reduces the processing time that is  
12 required to render a multi-media project. Step 3902 defines a first data structure  
13 that represents the editing project. Any suitable data structure can be utilized. In  
14 the present example, a data structure in the form of a hierarchical tree is utilized.  
15 An exemplary tree is shown in Fig. 30. Step 3904 processes the first data structure  
16 to provide a second data structure that is configured to program a matrix switch.  
17 In the illustrated example, the second data structure comprises a grid structure.  
18 Exemplary processing is described in the context of Figs. 30-37. Step 3906 then  
19 programs the matrix switch using the second data structure.

## 21 **Dynamic Graph Building**

22 Having introduced the various architectural and implementation elements  
23 of the present invention, above, attention is now drawn to Figs. 40-44, wherein  
24 another aspect of the illustrated embodiment is presented. As introduced above,  
25 each matrix switch filter 308 is time aware. That is, according to one

1 implementation, matrix switch 308 maintains one or both project time and media  
2 source time information. This enables the matrix switch 308 to, among other  
3 things, throttle delivery of media content to the matrix switch 308.

4 As an extension of this capability, in accordance with one aspect of the  
5 present invention, render engine 222, via matrix switch filter 308, dynamically  
6 builds a filter graph representation of a project during execution of the filter graph.  
7 That is, render engine 222 based, at least in part by the control performed by  
8 matrix switch 308, dynamically loads filter graph chains as they are needed.  
9 Further, render engine 222 may well discard, or cache processing chains when  
10 they are no longer required to support execution of the processing project. To  
11 illustrate the benefits afforded by dynamic graph building, assume, for example,  
12 that an editing project included over 100 sources, yet only three (3) of them were  
13 ever required at any given time to support execution of the filter graph. Those  
14 skilled in the art will appreciate that loading three sources will be executed much  
15 fast than 100 sources, thereby permitting execution of the filter graph to  
16 commence much more rapidly than conventional filter graph implementations.  
17 Further, the memory and processing resources required to support three (3) sources  
18 will generally be less than those required to support 100 sources. Thus, those  
19 skilled in the art will appreciate that the dynamic graph building properties of the  
20 present invention reduce the computational and memory requirements placed on  
21 the host system (e.g., computing system 200).

22 **Fig. 40** is a flow chart of an example method for processing media content,  
23 in accordance with one embodiment of the present invention. More particularly,  
24 Fig. 40 illustrates an example method wherein render engine 222 dynamically  
25 generates and manages a filter graph to reduce the computational and/or memory

1 requirements placed on a host system. As shown, method 4000 begins with block  
2 4002, wherein render engine 222 receives an indication to generate a development  
3 project. According to one implementation, as discussed above, render engine 222  
4 receives the indication from a higher-level application 216, e.g., media processing  
5 application 224, to assist a user in generating a processing project (e.g., a media  
6 processing project).

7 In block 4004, render engine 222 identifies the number and nature of the  
8 media sources within the user-defined processing project, in preparation for  
9 generating a filter graph representation of the processing project. As introduced  
10 above, for each of the identified sources, render engine 222 determines the  
11 necessary transform filters 306 required to pre-process the source (i.e., the source  
12 processing chain), preparing the chain for presentation to the matrix switch filter  
13 308 and one or more transition/effect filters 306. Unlike conventional  
14 implementations which would proceed to generate the entire filter graph in  
15 preparation for execution of the processing project, render engine 222 generates a  
16 list of sources and when they are required in the filter graph. An example of a data  
17 structure comprising a list of processing chains is presented with reference to Fig.  
18 41.

19 Turning briefly to **Fig. 41**, a graphical illustration of an example data  
20 structure comprising a processing chain execution list is presented. As shown, the  
21 chain execution list 4100 is comprised of a number of information fields, e.g.,  
22 4102-4110 which detail, in part, which chains are required at a particular time in  
23 project execution. In accordance with the illustrated example embodiment of Fig.  
24 41, chain execution list 4100 is depicted comprising a chain identifier field 4102, a  
25

1 source identifier field 4104, a project time field 4106, a source time field 4108,  
2 and a dependencies field 4110.

3 Upon identifying a project source and the associated filters required for pre-  
4 processing the source (i.e., the source chain), render engine 222 assigns the chain  
5 an identifier which uniquely identifies the source chain within the context of the  
6 filter graph. Accordingly, the chain execution list 4100 includes a field 4102  
7 which maintains a list of the chains utilized in the associated project. Within the  
8 context of the filter graph, the chain identifier corresponds to the source and the  
9 associated pre-processing filters.

10 The source identifier field 4104 contains information denoting the project  
11 source associated with a particular chain identifier. In this regard, the source  
12 identifier field 4104 may well contain a file name, a file handle, or any other  
13 suitable source identifier.

14 The project time field 4106 denotes at what point during project execution  
15 the source chain is required. The source time field 4108 denotes what portion of  
16 the source file is required to support execution of the processing project. It should  
17 be appreciated that a user may well utilize the whole source file or any part  
18 thereof, as defined by the processing project.

19 The dependencies field 4110 denotes whether the associated chain is  
20 dependent upon any other chain. As will be described in greater detail below,  
21 multiple chains may rely on a common source and/or a subset of another source  
22 chain. In certain implementations, it would not be advantageous to unload source  
23 chains prior to their execution and/or the execution of chains dependent thereon.  
24 Accordingly, render engine 222 maintains a list of such dependencies within the  
25 chain execution list 4100. It is to be appreciated, however, that certain

1 circumstances may arise where it is necessary to unload a chain prior to or during  
2 execution, or prior to execution of an otherwise dependent chain. One such  
3 example is where the processing project utilizes a hierarchical structure, wherein  
4 individual chains are assigned a priority level. An implementation is  
5 contemplated, for example, wherein the priority of a particular chain is  
6 dynamically managed by a matrix switch filter 308 within a filter graph based, at  
7 least in part, on how soon the chain is required to support the uninterrupted  
8 execution of the processing project, i.e., chains which are required more urgently  
9 are assigned a higher priority and, as a result, are processed at the disadvantage of  
10 other, lower priority chains. In the extreme, lower priority chains are unloaded to  
11 enable loading of a higher priority chain. It is to be appreciated that, although  
12 depicted as a two-dimensional data structure, chain execution lists of greater or  
13 lesser complexity may well be substituted without deviating from the spirit and  
14 scope of the present invention.

15         Returning to Fig 40 and, in particular, block 4006, render engine 222  
16 dynamically generates and manages a filter graph representation of the processing  
17 project invoking only those chains associated with sources that are necessary to  
18 support the current and/or impending execution of the filter graph. It is to be  
19 appreciated that by not opening each of the chains of a processing project, render  
20 engine 222 reduces the amount of memory required to build the filter graph,  
21 thereby reducing the amount of memory required to complete execution of the  
22 project, i.e., recall the example where the entire graph utilized 100 sources, but  
23 only required three (3) at any given time. An example method of dynamically  
24 generating and managing a filter graph is presented with reference to the flow  
25 chart illustrated in Fig. 42.



Turning to **Fig. 42**, an example method for dynamically generating and managing a filter graph is presented, in accordance with one embodiment of the present invention. In accordance with the illustrated example implementation of Fig. 42, method 4006 commences with block 4202 wherein render engine 222 determines which chains are required to fulfill execution of the development project for the next M seconds. According to this example implementation, M must be greater than the minimum time it takes to completely load the next chain. In accordance with the illustrated example implementation, wherein matrix switch filter 308 controls the pace of project execution, matrix switch filter 308 provides an indication to render engine 222 of what chains are required in block 4202.

According to one implementation, M is dynamically generated based on a number of factors including, but not limited to, processing speed, available memory, the complexity of the development project, the number and type of the source chains, and the like. In certain implementations, processing system 300 maintains a performance history (not shown), and dynamically modifies the processing threshold M based on past performance. According to one implementation, M is stochastically set to ten (10) seconds. Accordingly, render engine 222 maintains only the chains currently required to support the next ten seconds of execution. It is important to note that project execution does not necessarily correlate to rendering of the composite generated by the filter graph. That is, in certain implementations, execution of the filter graph is performed as fast as possible, utilizing the shared memory resources of the matrix switch filter 308 to buffer the composite until the rendering chain consumes the composite.

In block 4204, render engine 222 determines whether a threshold of loaded chains (e.g., a maximum chain-count) (T) has been exceeded. In certain

1 implementations, the number of loaded chains will be limited due to memory  
2 limitations. According to one example implementation, setting T equal to one (1)  
3 is popular in that it requires the render engine to analyze the filter graph for chains  
4 that are no longer required (e.g., exhausted chains) whenever a new chain is  
5 considered for loading. According to one example implementation, the maximum  
6 number of loaded chains supported by render engine 222 is seventy (70).  
7 Accordingly, once render engine 222 has identified the chains required (block  
8 4202), a determination is made of whether there is space in which to load them  
9 into the filter graph (block 4204).

10 If the chain count threshold (T) has not yet been reached, render engine 222  
11 loads the identified chains, block 4206. Matrix switch filter 308 will initiate  
12 execution of the newly loaded chains to fulfill the execution requirements of the  
13 development project.

14 If, in block 4204 the chain-count threshold (T) has been reached, render  
15 engine 222 determines whether one or more chains may be unloaded from the  
16 filter graph. Thus, in block 4206, render engine 222 identifies any currently loaded  
17 chains that will not be utilized in the next N seconds. Source chains may be  
18 accessed multiple times to process multiple portions of an associated source.  
19 Thus, in accordance with steps 4202-4206, a source chain may have been loaded  
20 to meet an impending execution requirement, and remains loaded to satisfy a  
21 subsequent processing task. However, where resources are running short, render  
22 engine 222 along with matrix switch filter(s) 308 determine which chains are not  
23 required in the next N seconds and, in block 4210, instructs render engine 222 to  
24 unload the identified chains. As above, N may well be dynamically derived based  
25 on past performance. In accordance with one example implementation, N is thirty

(30) seconds. According to one implementation, render engine 222 determines whether the chains will be required for subsequent processing in the current or a future filter graph. If so, the filter chain is removed from the active filter graph by render engine 222 and cached for subsequent re-integration in this or a future filter graph.

In block 4212, render engine 222 determines whether unloading of the identified chain(s) in block 4210 has brought the total chain-count below the threshold a cutoff threshold (V). According to one implementation, V is greater than T. This is particularly useful if T has been set to one (1), as described above. If so, processing continues with block 4206 as render engine 222 loads the chains identified in block 4202.

If, in block 4212, the chain-count threshold is still exceeded, render engine 222 re-analyzes the current filter graph and identifies the lowest priority chains, block 4214. That is, the filter graph may well be comprised of seventy chains, all of which will be required in the next thirty (N) seconds. If, however, the chains identified in block 4202 are needed prior to any of the seventy chains currently loaded in the filter graph, those chains are assigned a lower priority. Processing continues with block 4210 as the lower priority chains are unloaded, as render engine 222 re-analyzes the chain-count, in block 4212. If the filter graph has space available, processing continues with block 4206, else it continues with block 4214.

**Fig. 43** graphically illustrates an example data structure utilized to manage dynamic graph building, according to one example implementation. In accordance with the illustrated example embodiment of Fig. 43, a filter graph 4300 is presented. Unlike conventional filter graph implementations, wherein all chains

1 4302-4308 would be loaded prior to execution of the development project, filter  
2 graph 4300 illustrates the dynamic nature of the present invention. In the  
3 illustrated example of Fig. 43, matrix switch filter 308 has identified at least two  
4 source chains 4302, 4304 which are required in the next M seconds to support the  
5 timely processing of the development project. Such chains are illustrated in Fig.  
6 43 with a solid black line to denote that these chains are currently loaded into the  
7 filter graph 4300. In accordance with one aspect of the present invention, the  
8 development project may well contain additional chains (e.g., 4306 and 4308) that  
9 will be required to complete execution of the development project, but which are  
10 not yet required and are, thus, not yet loaded. Such chains are illustrated in Fig. 43  
11 with dotted lines, denoting that they are not currently loaded into the filter graph  
12 4300. By limiting the number of currently loaded chains to a threshold (T) or,  
13 alternatively (V), the present implementation reduces the memory requirements  
14 necessary to satisfy even the most complex of development projects by unloading  
15 chains when they are no longer required, without stifling the user's creativity by  
16 artificially limiting the size of the filter graph.

17 **Fig. 44** is an example filter graph denoting chain dependencies. In  
18 accordance with the illustrated example of Fig. 44, filter graph 4400 depicts two  
19 chains 4402 and 4404, each coupled to an associated matrix switch filter through  
20 an innovative parser object described more fully in a co-pending patent application  
21 entitled *A System and Related Methods for Reducing the Instances of Source Files*  
22 *in a Filter Graph*, filed on December 6, 2000 by the inventors of the present  
23 application, the disclosure of which is hereby incorporated by reference.

24 Filter graph 4400 is representative of the situation wherein the source is a  
25 multimedia file containing both audio and video content, each requiring a

1 dedicated pre-processing chain and a matrix switch 308. It is to be appreciated  
2 that in such a situation, where the video chain 4202 and the audio chain 4204 share  
3 a single instance of a source filter, a dependent relationship with respect to  
4 unloading the source filter chain. In such a circumstance, it might not be desirable  
5 for the video matrix switch, for example, to unload chain 4202 upon completion if,  
6 for example, the audio matrix switch requires additional content from the source.  
7 Thus, a conventional approach to such a situation would be to invoke separate  
8 processing chains, one for the audio content and one for the video content, each  
9 having a unique instance of the source filter. The conventional approach has the  
10 disadvantage of wasting memory by invoking multiple instances of the same  
11 source. To alleviate this problem, the render engine 222 identifies such  
12 dependencies within the chain execution list and will not seek to unload the  
13 source filter until both processing chains no longer require content from the source  
14 (e.g., for at least the next N seconds, as introduced above). In this regard, the  
15 present invention alleviates the need to construct two complete chains accessing a  
16 common source, thereby reducing the memory requirements necessary to support  
17 more complex processing projects.

## 19 **Caching of Processing Chains**

20 Attention is now directed to Figs. 45-49, wherein another aspect of the  
21 illustrated embodiment is presented. As introduced above, conventional  
22 implementations of the filter graph manager required a source processing chain be  
23 constructed for each access to a source. Thus, a literal implementation of the  
24 dynamic graph building feature introduced above might well have the adverse  
25 affect of requiring that multiple accesses to a source would require that a

1 commensurate number of processing chains be constructed, i.e., one for each time  
2 the filter string was dynamically added to the filter graph. As introduced above,  
3 performance improvements may be achieved by reducing the number of times a  
4 processing chain is created to retrieve media content from a particular source.  
5 Accordingly, a system and method of caching a string of connected filters is  
6 introduced in association with Figs. 45-48. Moreover, as will be developed more  
7 fully below, render engine 222 beneficially caches processing chains for  
8 subsequent use within a development project (i.e., later in the execution of the  
9 filter graph) and for use across development projects. It is to be appreciated,  
10 however, that the following is but one example implementation of the broader  
11 inventive concept of caching processing chains for use in filter graph processing.  
12 Alternative methods of greater or lesser complexity may well be used within the  
13 spirit and scope of the present invention. Indeed, such alternative methods are  
14 anticipated within the scope of the present invention.

15 **Fig. 45** is a block diagram of an example render engine 222 incorporating  
16 storage space for filter strings, in accordance with one aspect of the present  
17 invention. In accordance with the illustrated example embodiment of Fig. 45, a  
18 block diagram of render engine 222 depicting a development project including  
19 source processing chains 4504, 4506 and 4508 coupled to dynamic matrix switch  
20 filter 308 is presented. In addition, render engine 222 is depicted comprising a  
21 processing chain cache 4502. According to one example implementation, render  
22 engine 222 includes a processing chain cache 4502 within which render engine  
23 222 can store whole filter strings for subsequent use in execution of a development  
24 project. That is, rather than unload a source filter string (e.g., 4504) once  
25 processing of a first subset of media content from the source has been completed,

1 innovative render engine 222 determines whether it is subsequently needed (1) to  
2 support subsequent execution of the current development project, or (2) to support  
3 execution of a subsequent development project. According to one implementation,  
4 if render engine 222 determines that the current and/or subsequent development  
5 project will use the processing chain, or a modified version of the processing  
6 chain, render engine 222 caches the source processing chain in processing chain  
7 cache 4502 for later retrieval and reintegration with a development project.  
8 According to another implementation, render engine 222 assumes that processing  
9 chains will be subsequently required, and caches all chains upon removal from a  
10 development project, employing standard cache management when the cache  
11 becomes full. Accordingly, render engine 222 reduces the time and processing  
12 resources that would otherwise be spent in continuously recreating a source  
13 processing chain from scratch each time it is required by the render engine 222.

14 Although processing chain cache 4502 is depicted as being integrated  
15 within render engine 222, it is to be appreciative that such a configuration is for  
16 ease of explanation only. That is, those skilled in the art will appreciate that  
17 processing chain cache 4502 is some portion of a host's system memory (e.g.,  
18 system memory 204) and, as a result, may well reside external to the render engine  
19 222 and, for that matter, external to the host itself. Thus, the illustrated  
20 embodiment is to be considered but an example of the broader inventive concept  
21 of the present invention.

22 **Fig. 46** is a graphical illustration of an example processing chain cache  
23 suitable for use in accordance with the teachings of the present invention. In  
24 accordance with the illustrated example embodiment of Fig. 46, an example data  
25 structure comprising a processing chain cache 4502 is graphically presented. As

1 shown, processing chain cache 4502 includes at least two fields, a processing  
2 chain identifier field 4602 and a field within which to store the connected  
3 processing chains 4604.

4 According to one implementation, to be described more fully below, to  
5 facilitate the swift and accurate identification and retrieval of processing chains,  
6 each chain is assigned a unique string identifier before being cached in processing  
7 chain cache 4502. In accordance with the illustrated example embodiment of Fig.  
8 46, a numeric identifier, e.g., 102, 205 and 377, is employed to particularly  
9 identify processing chains 4504, 4506 and 4508, respectively. In other  
10 embodiments, an identifier uniquely associated with the source (e.g., a source  
11 handle, file name, etc.) is used as the processing chain identifier. Subsequently,  
12 when render engine 222 needs to add a processing chain to a development project,  
13 it first accesses the processing chain cache 4502 to determine whether the  
14 processing chain is available from within the cache, before constructing the source  
15 processing chain from scratch. According to one implementation, if render engine  
16 222 identifies the required source (e.g., by source handle), but the processing chain  
17 does not meet the current requirements, render engine 222 pulls the identified  
18 chain from cache 4502 for integrated with the development project, and then  
19 modifies the extracted processing chain in accordance with the requirements of the  
20 development project. Those skilled in the art will appreciate that it is often a faster  
21 operation to pull an existing processing chain from cache 4502 and modify it than  
22 it would be to recreate the processing chain from scratch. In this regard, render  
23 engine 222 employs the innovative processing chain cache 4502 to improve  
24 performance characteristics of the development system.



1       The processing chain field 4604 stores a pointer to a memory location  
2 wherein the processing chain objects are stored. Although depicted in graphical  
3 form, it is to be appreciated that the processing chains are comprised of software  
4 objects (e.g., COM objects) and are, therefore, readily storable within a typical  
5 memory space. Thus, the graphical representation in Fig. 46 is for ease of  
6 understanding only. Although depicted as a two-dimensional data structure, those  
7 skilled in the art will appreciate that caches of greater or lesser complexity may  
8 well be employed as processing chain cache 4502 without deviating from the spirit  
9 and scope of the present invention.

10       Turning to **Fig. 47**, a flow chart of an example method for dynamically  
11 loading processing chains is presented, according to one aspect of the invention.  
12 In accordance with the illustrated example embodiment of Fig. 47, an example  
13 method of dynamically loading processing chains (block 4206 of Fig. 42) begins  
14 with block 4702 wherein render engine 222 identifies the sources to be used  
15 during execution of a next section (e.g., M seconds) of the development project.

16       In block 4704, render engine 222 determines whether the source(s) have  
17 been previously used in this, or a prior development project. According to one  
18 implementation, render engine 222 maintains a log of accessed source handles  
19 associated with previously used sources, and consults the log to determine whether  
20 the source has been previously accessed. In an alternate implementation, render  
21 engine 222 accesses one or more processing chain caches to determine whether  
22 there is a hit, denoting that the source has previously been used.

23       If, in block 4704, render engine 222 determines that the source has, indeed,  
24 been accessed before, render engine 222 issues a request including at least the  
25 processing chain identifier to one or more processing chain cache(s) 4502. More

1 particularly, render engine 222 issues a request to retrieve a source processing  
2 chain associated with a particular identifier from the one or more cache(s) 4502.  
3 In block 4708, render engine 222 determines whether there was a "hit" to the  
4 request in the cache(s) 4502 and, if so, render engine 222 loads the identified  
5 source processing chain from one or more memory location denoted by the pointer  
6 in the processing chain cache 4502, block 4710.

7 According to one aspect of the embodiment, alluded to above, render  
8 engine 222 may not find an exact match in the processing chain cache 4502. That  
9 is, render engine 222 may well find a source processing chain with the requisite  
10 source, but the processing chain is not accurate for the required implementation.  
11 In such a case, depending on the modifications to be made, render engine 222 may  
12 well call the cached processing chain for integration in the development project,  
13 and modify the processing chain as necessary, in accordance with the requirements  
14 of the development project, block 4711. In this regard, render engine 222 may  
15 well add, change or remove individual processing objects (e.g., filters) from the  
16 processing chain to satisfy the requirements of the development project.

17 In block 4712, render engine 222 updates the active development project  
18 with the retrieved source processing chain in support of subsequent execution of  
19 the development project block 4712.

20 However, if in block 4708 the request is not successful in returning the  
21 requested processing chain, or if in block 4704 the render engine 222 determines  
22 that the source has not been previously accessed, render engine 222 must assemble  
23 the source processing chain from scratch. Accordingly, the process continues with  
24 block 4714, wherein render engine 222 identifies the processing objects required  
25 to appropriately pre-process the media content provided by the source, in

1 accordance with the development project definitions. Once identified, render  
2 engine 222 assembles the source processing chain necessary to support media  
3 processing defined by the development project, block 4716. Once assembled,  
4 render engine 222 updates the development project with the source processing  
5 chain in support of continued execution of the development project, block 4712.

6 **Fig. 48** is a flow chart of an example method for dynamically unloading a  
7 processing chain from a development project, according to one embodiment of the  
8 present invention. As shown, the method of Fig. 48 begins with block 4802  
9 wherein render engine 222 determines whether a source processing chain will be  
10 required in subsequent processing of the development project. More particularly,  
11 render engine 222 determines whether the development project requires additional  
12 media content from the source. If, in block 4804, render engine 222 determines  
13 that the source processing chain is not required in subsequent execution of the  
14 project, render engine further determines whether the chain might be required in  
15 future projects. According to one implementation, render engine 222 assumes that  
16 all processing chains may well be required in future processing projects, wherein  
17 the process continues with block 4812 (described more fully below).

18 If, however, render engine 222 determines that the processing chain will  
19 not be required in future execution of the current or subsequent projects, render  
20 engine 222 concludes not to cache the processing chain 4808, wherein render  
21 engine 222 unloads the processing chain, block 4810.

22 If, in blocks 4804 or 4808, render engine 222 decides that the processing  
23 chain may well be used subsequently, render engine 222 assigns a unique  
24 processing chain identifier to the chain, block 4812. As introduced above, the  
25 processing chain identifier may take many alternate forms such as, e.g., a

1 numerical identifier, a source file handle, an alphanumeric identifier, and the like.  
2 In block 4814, the a pointer to the processing chain denoted by the unique  
3 identifier is stored in one or more processing chain cache(s) 4502 for subsequent  
4 retrieval.

5 It is to be appreciated, given the foregoing, that use of the processing chain  
6 cache 4502 facilitates performance improvements in the initial loading of a  
7 development project as well as the dynamic graph building features of the  
8 innovative render engine 222, described above. More particularly, the processing  
9 chain cache 4502 enables render engine 222 to temporarily remove currently  
10 inactive processing chains from a development project, while retaining the ability  
11 to immediately recall and reintroduce them into the project without having to re-  
12 create the processing chain from scratch.

13 Although the invention has been described in language specific to structural  
14 features and/or methodological steps, it is to be understood that the invention  
15 defined in the appended claims is not necessarily limited to the specific features or  
16 steps described. Rather, the specific features and steps are disclosed as preferred  
17 forms of implementing the claimed invention.